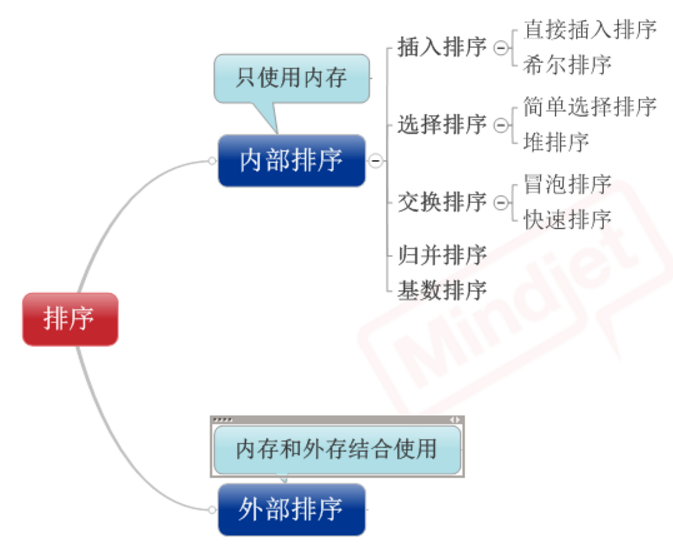
# 常用排序算法



## 插入排序

#### 算法描述

插入排序是在一个已经有序的小序列的基础上，一次插入一个元素。当然，刚开始这个有序的小序列只有1个元素，就是第1个元素。比较是从有序的序列末尾开始，也就是想要插入的元素和已经有序的最大者开始比较，如果比它大则直接插入在其后面，否则一直往前找直到找到它该插入的位置。如果碰见一个和插入元素相等的，那么插入元素就要放在相等元素的后面。所以相等元素的前后顺序没有改变，所以排序是稳定的。

具体描述：

1. 从第一个元素开始，该元素可以认为已经被排序
2. 取出下一个元素，在已经排序的元素序列中从后向前扫描
3. 如果该元素(已排序)大于新元素，将该元素移到下一位置
4. 重复步骤3，直到找到已排序的元素小于或者等于新元素的位置
5. 将新元素插入到该位置
6. 重复步骤2~5

如果目标是把n个元素的序列升序排列，那么采用插入排序存在最好情况和最坏情况。最好情况就是，序列已经是升序排列了，这样需要进行的比较操作是(n-1)次。最坏情况是，序列是降序排列，那么此时需要进行的比较是n(n-1)/2次。

平均来说插入排序算法时间复杂度为O(n^2)，因此插入排序不适合对于数据量比较大的排序应用。但是如果数据量比较小，例如，量级小于千，那么插入排序还是一个不错的选择。

时间复杂度：最好情况(已经排好序的情况)O(n)，最坏情况(与所要排顺序相反)O(n^2)

#### 考察点

基本在笔试选择填空问时间复杂度时才可能出现。毕竟排序速度比较慢，因此在算法大题中考察的速度比较少。

## 二、二分插入排序

#### 算法简介

二分(折半)插入排序是一种在直接插入排序算法上进行小改动的排序算法。其与直接插入排序算法最大的区别是查找插入位置时使用的是二分查找的方式。在速度上有一定提升。

#### 算法描述和分析

1. 从第一个元素开始，该元素可以认为已经被排序
2. 取出下一个元素，在已经排序的元素序列中二分查找到第一个比它大的数的位置
3. 将新元素插入到该位置后
4. 重复上述步骤
5. 稳定
6. 空间代价：O(1)
7. 时间代价：最佳情况O(nlogn)，最差和平均情况O(n^2)

## 考察点

这个排序算法出现的频率也不高，但毕竟是直接排序算法的一个小改进算法，同时二分查找又是很好的思想，可能会在面试的时候提到。

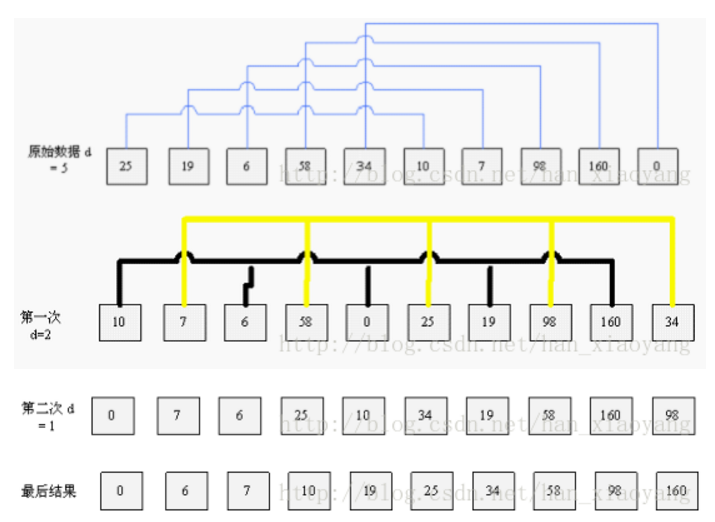
## 三、希尔排序

希尔排序，也称递减增量排序算法，是插入排序的一种高速而稳定的改进版本。基本思想是将相距某个“增量”的记录组成一个子序列，对每个子序列进行直接插入排序，然后减小“增量”，再对每个子序列进行直接插入排序，以此循环，直到“增量”为1，然后进行最后直接插入排序。

增量间隔序列函数h = h\*3+1

希尔排序比插入排序快很多的原因：当h值很大时，数据项每一趟排序移动的元素个数少，但移动的距离很长，这是非常高效的；当h值减小时，每一趟排序移动的元素个数增加，但此时的数据项已经接近于他们最终排序后的位置，插入排序可以更有效。

#### 算法图解



最坏情况时间复杂度：O(n^2)

选择的步长和时间复杂度的关系

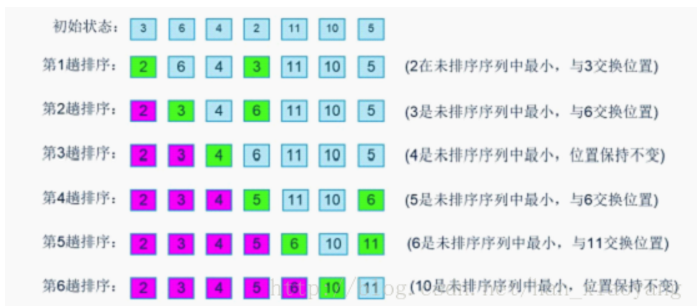
## 四、选择排序

#### 算法简介

首先在未排序序列中找到最小(大)元素，与排序序列的起始位置元素交换，然后，再从剩余未排序元素中继续寻找最小(大)元素，然后与已排序序列元素后面的元素进行交换。以此类推，直到所有元素均排序完成。

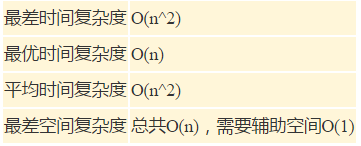


#### 算法图解

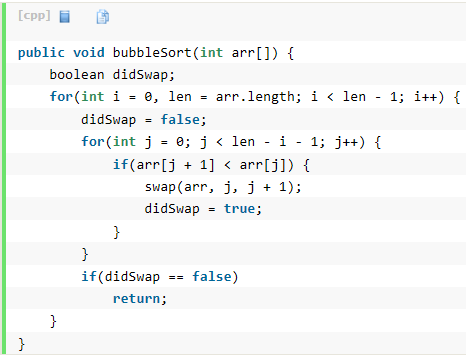


## 五、冒泡排序

#### 时间复杂度



这里要说下为什么最优时间复杂度是O(n)。因为之前的O(n＾2)是一种通用写法，并没考虑最好的情况，即已经排好序。可以在外层遍历的过程当中设置一个标志位为false，如果在循环的过程中没有发生交换，说明已经排好序，那么标志位为false，如果发生了交换那么标志位就为true，这样在后面判断为true时直接退出。



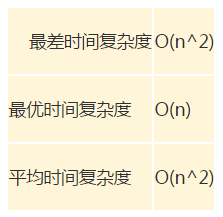
#### 考点

冒泡排序是一个很常见的考点，平均时间空间复杂度，最好最坏情况下的时间复杂度，在不同情况下每一趟的比较次数，以及加标志位减少比较次数等，都是需要注意的地方。

## 六、鸡尾酒排序/双向冒泡排序

第一趟从a到b冒泡，到了b再逆序冒泡到a，第二趟重复第一趟的情况

时间复杂度：



出现频率不高，选择填空会少量出现，并且多以双向冒泡排序出现。

## 七、快速排序

#### 算法简介

通过一趟排序将待排记录分隔成独立的两部分，其中一部分记录的关键字均比另一部分的关键字小，则可分别对这两部分记录继续进行排序，以达到整个队列有序。

#### 步骤

1. 从数列中调出一个元素称为基准
2. 所有元素比基准值小的摆放在基准前面，比基准大的摆放在基准后面(相同的数可以到任一边)。在这个分区退出之后，该基准就处于数列的中间位置。
3. 递归的把左子列和右子列排序



## 八、堆排序

堆排序很容易出现，选择填空问答算法大题都会出现，建堆的过程，堆调整的过程，这些过程的时间复杂度，空间复杂度，以及如何应用在海量数据topK问题中。

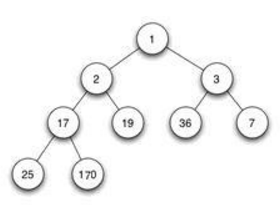
#### 算法简介

堆积是一个近似完全二叉树的结构，并同时满足堆积的性质；即子节点的键值或索引总是小于(或者大于)它的父节点。

我们这里指的堆一般都是指的二叉树，它满足两个特性：

1. 父节点的键值总是大于或等于(小于或等于)任何一个子节点的键值
2. 每个节点的左子树或右子树都是一个二叉堆(都是最大推或最小堆)。

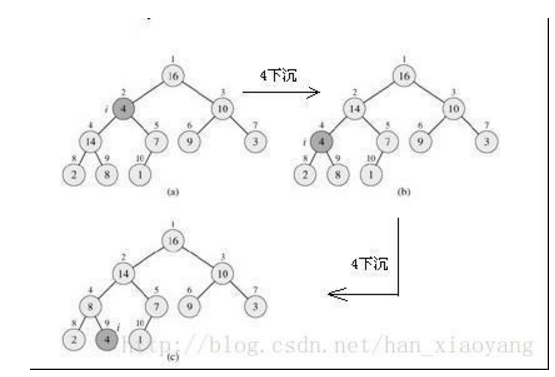
如下为一个最小堆：



堆调整—

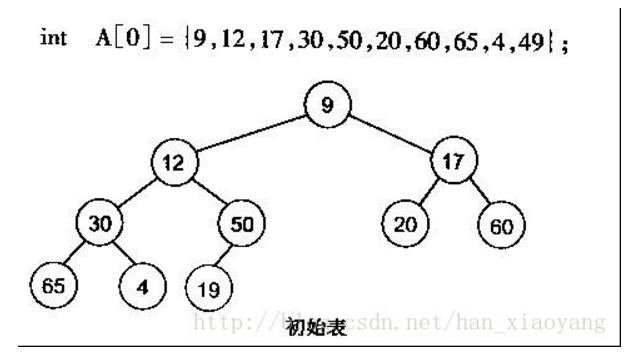
这个为了保证堆的特性而做的一个操作。对某一个节点为根的子树做堆调整，其实就是将该根节点进行“下沉”操作(具体是通过和子节点交换完成的)，一直下沉到合适的位置，使得刚才的子树满足堆的性质。

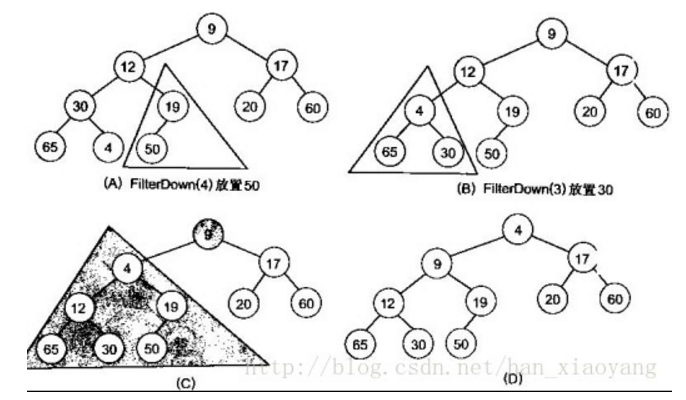
做一次堆调整的时间复杂度为log(n)



如何建堆—

建堆是一个通过不断的堆调整，使得整个二叉树中的数满足堆性质的操作。在数组中的话，我们一般从下标为n/2的数开始做堆调整，一直到下标为0的数(因为下标大于n/2的数都是叶子节点，其子树已经满足堆的性质了)。下图为一个图示：





如何进行堆排序

## 九、归并排序

## 十、桶排序

## 十一、计数排序

## 十二、基数排序